Abalone Fish Age Estimation with Support Vector Machines

August 18, 2019

1 Abalone Fish Age Estimation with Support Vector Machines

In this project, I build a support vector machine classifier and regression models to predict the age of an Abalone fish based on their physical measurements. The age of the fish is denoted by the number of rings on this fish. I start by cleaning and preparing the data set, then I take my best guess for the hyperparameters and kernel to use in the support vector classifier (SVC). Next, I'll test out different kernels and hyper parameters to fine tune the SVC.

I'm curious about how the results will be affected if I keep the rings variable continuous and employ a support vector regression (SVR). I'll take a similar approach as I did to the SVC and evaluate the two models by comparing their recall, precision, and f-measures.

2 Import, clean, and prepare data

To begin, I import the Abalone fish data and take a look at the first five rows and the summary statistics of the dataset. In conjunction with the dataset description, I can see that the sex variable is made up of three different categorical variables (male, female, and infant). I start by one-hot encoding this column to generate three additional binary columns to denote the sex of the fish. Next, in order to use the SVC, the target variable needs to be binary. I encode the rings variable based on the average number of rings in the dataset (about 10 rings). Therefore, I can predict whether the fish is above or below the average age of the population using the SVC. If the fish is above 10 rings, the data is encoded as a 1 and if the fish is below 10 rings, it's encoded as a 0.

Now that the data has been prepared for the SVC, I split the dataset into testing and training subsets to be fed into the model.

```
In [1]: # import fish data and column names
        import pandas as pd
        data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.d

        cols = ['sex', 'length', 'diameter', 'height', 'whole_weight', 'shuck_weight', 'viscera
               'shell_weight', 'rings']

        fish = pd.read_csv(data_url, names=cols)  # read in data, set column names, and set NA

        fish.head()
```

```
Out[1]:   sex  length  diameter  height  whole_weight  shuck_weight  viscera_weight  \
        0   M   0.455     0.365   0.095        0.5140        0.2245          0.1010
```

```
1   M   0.350     0.265   0.090        0.2255        0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770        0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160        0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050        0.0895          0.0395

   shell_weight  rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7
```

In [2]:
```
fish.describe()
```

Out[2]:
```
             length     diameter       height  whole_weight  shuck_weight  \
count  4177.000000  4177.000000  4177.000000   4177.000000   4177.000000
mean      0.523992     0.407881     0.139516      0.828742      0.359367
std       0.120093     0.099240     0.041827      0.490389      0.221963
min       0.075000     0.055000     0.000000      0.002000      0.001000
25%       0.450000     0.350000     0.115000      0.441500      0.186000
50%       0.545000     0.425000     0.140000      0.799500      0.336000
75%       0.615000     0.480000     0.165000      1.153000      0.502000
max       0.815000     0.650000     1.130000      2.825500      1.488000

       viscera_weight  shell_weight        rings
count     4177.000000   4177.000000  4177.000000
mean         0.180594      0.238831     9.933684
std          0.109614      0.139203     3.224169
min          0.000500      0.001500     1.000000
25%          0.093500      0.130000     8.000000
50%          0.171000      0.234000     9.000000
75%          0.253000      0.329000    11.000000
max          0.760000      1.005000    29.000000
```

In [3]:
```
# one hot encode the sex column
encode = pd.get_dummies(fish['sex'])
encoded_fish = pd.concat([fish, encode], axis = 1)
# drop the now redundant sex column
encoded_fish = encoded_fish.drop(['sex'], axis = 1)
# rename the encoded columns to be more descriptive
encoded_fish = encoded_fish.rename(columns = {'M':'male',
                                              'F':'female',
                                              'I':'infant'})
```

In [4]:
```
# encode the rings data so that 1 represents if the rings are above the average and 0
encoded_fish.loc[:, 'above_avg_age'] = (encoded_fish.loc[:, 'rings'] >= 10).astype(int)
```

In [5]:
```
# define the features and the targets for classification
X = encoded_fish.drop(['rings', 'above_avg_age'], axis = 1)
Y = encoded_fish['above_avg_age']
```

```
In [6]: from sklearn.model_selection import train_test_split

        # split into a training and testing set
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=
```

3 SVC

Prior to instantiating the SVC, I took a guess at a few hyper parameter values. The cost parameter represents the level of penalty to give to the error term, in this case I selected 0.9 which is quite stringent. This means that I'm imposing a higher cost for misclassifications and employing what is called a "hard margin". If the cost value was lower, it would allow the model more leeway and provide a "soft margin". Soft margins tend to be more general and have a lower sensitvity for noise. The gamma parameter defines how influential the feature differences are in the prediction. I set gamma to be equal to 5 to start.

After instantiating the model and fitting to the training data, I obtained the predicted target from the test features and compared the predictions to the test target. Using the classification report, we can evaluate how the model performed by looking at the precision, recall, and f1-score. I will be primarily looking at the f1-score since it combines precision and recall. This model performed reasonably well with an f1-score of 0.76. I'll try to improve this score by tweaking the hyperparameters and trying other kernels.

```
In [7]: # take a best guess at the hyper parameters to use
        cost = .9 # penalty parameter of the error term
        gamma = 5 # defines the influence of input vectors on the margins

In [8]: from sklearn import svm, metrics
        from sklearn.metrics import classification_report

        # test a LinearSVC with the initial hyper parameters
        clf1 = svm.LinearSVC(C=cost).fit(X_train, y_train)
        clf1.predict(X_test)
        print("LinearSVC")
        print(classification_report(clf1.predict(X_test), y_test))
```

```
LinearSVC
              precision    recall  f1-score   support

           0       0.76      0.76      0.76       416
           1       0.76      0.76      0.76       420

   micro avg       0.76      0.76      0.76       836
   macro avg       0.76      0.76      0.76       836
weighted avg       0.76      0.76      0.76       836
```

4 Testing other kernels in the SVC

There is a "kernel trick" you can employ when fine tuning SVCs. Ideally you want to use a linear kernel when you data is linear. However, if you have data that is too non-linear, you can transform your input variables so that the shape of your dataset becomes more linear. To do this you can select different kernels to transform your dataset.

In this case, I tried the rbf and poly kernels to see if they perform better than the linear kernel above. Based on the f1-score, both the rbf and poly kernels perform slightly better than the linear kernel with f1-scores of 0.77 and 0.78, respectively.

```
In [9]: # test linear, rbf and poly kernels
        for k in ('rbf', 'poly'):
            clf = svm.SVC(gamma=gamma, kernel=k, C=cost).fit(X_train, y_train)
            clf.predict(X_test)
            print(k)
            print(classification_report(clf.predict(X_test), y_test))
```

```
rbf
              precision    recall  f1-score   support

           0       0.71      0.80      0.75       368
           1       0.83      0.74      0.78       468

   micro avg       0.77      0.77      0.77       836
   macro avg       0.77      0.77      0.77       836
weighted avg       0.77      0.77      0.77       836

poly
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       393
           1       0.80      0.76      0.78       443

   micro avg       0.78      0.78      0.78       836
   macro avg       0.78      0.78      0.78       836
weighted avg       0.78      0.78      0.78       836
```

5 Fine tuning the SVC hyper parameters

I decided to move forward with the poly kernel to try and tweak some of the other hyper parameters in hopes of improving the model further. I updated the cost hyper parameter to be a bit more lienient (0.6) since we are only predicting if a fish is above or below the average age we don't need an extremely hard margin for our SVC. However, the f1-score does not change much even when reducing the cost hyper parameter. It seems like the best model in this case is an SVC with a poly kernel and a cost of 0.9.

```
In [10]: # poly
         poly = svm.SVC(gamma=gamma, kernel='poly', C=0.6).fit(X_train, y_train)
         poly.predict(X_test)
         print("poly")
         print(classification_report(poly.predict(X_test), y_test))
```

```
poly
              precision    recall  f1-score   support

           0       0.76      0.79      0.78       400
           1       0.80      0.77      0.79       436

   micro avg       0.78      0.78      0.78       836
   macro avg       0.78      0.78      0.78       836
weighted avg       0.78      0.78      0.78       836
```

6 Support Vector Regression

I was curious to see how well a support vector regression (SVR) model would predict the number of rings of the Abalone fish if we kept the number of rings continuous. I defined new targets and features for the SVR, the features were actually the same but I made a new variable for the regression for consistency. The target is now the continuous rings column that denotes the age of the Abalone fish. Next, I split the dataset into training and testing subsets.

I instantiated the SVR with a poly kernel and the same hyper parameters I tuned above in the SVC. To evaluate the SVR, I predict the Abalone rings based on the test features and then compare the predictions to the actual target values in the test dataset. The following metrics are employed to understand how well the model is performing: R-squared value, mean squared error, and the variance.

The R-squared value is pretty low, only 0.53, meaning that the model does not correlate the features with the number of rings very well. The mean squared error measures the average difference in error between the predicted values and the actual values. In this case, the mean squared error is about 5, ideally we'd want this value to be closer to 0 but it's still quite low which is a good sign. The variance score describes how far the actual values differ from the predicted values mean. This SVR has a variance score of 0.54 which is quite good, since we want a low variance to show that our predicted values are not deviating too far from the actual values.

```
In [11]: # define the features and the targets for regression
         X_reg = encoded_fish.drop(['rings', 'above_avg_age'], axis = 1)
         Y_reg = encoded_fish['rings']
```

```
In [12]: # split into a training and testing set
         X_regtrain, X_regtest, y_regtrain, y_regtest = train_test_split(X_reg, Y_reg, test_si
```

```
In [13]: from sklearn.metrics import *
         svr = svm.SVR(gamma = gamma, kernel = 'poly', C=0.8).fit(X_regtrain, y_regtrain)
         pred = svr.predict(X_regtest)
```

```
print("svr results:")
print("R-squared: ", svr.score(X_regtest, y_regtest))
print("Mean Squared Error: ", mean_squared_error(y_regtest, pred))
print("Variance: ", explained_variance_score(y_regtest, pred))
```

```
svr results:
R-squared:  0.5279085685031117
Mean Squared Error:  4.963693231922028
Variance:  0.5364710785202282
```

7 Conclusions

In this assignment I built a SVC and a SVR to predict the number of rings an Abalone fish has (the number of rings represent their age). The SVC was employed to predict whether the fish had rings greater than or less than the average number of rings in the dataset (10 rings). I fine tuned the hyper parameters and tried out a few different kernels to get the best model. I was curious to see how an SVR would perform in predicting the number of rings as a continuous variable. My findings and observations for each step are listed below:

Clean and prepare data: * To start, I one-hot encoded the "sex" column to create binary columns that hold the gender of each fish as either a male, female, or infant. * Next, I binned the "rings" column to a new column called "above_avg_rings". This column held a 1 where the number of rings was greater than 10 and a 0 where the number of rings was less than 10. * To prepare the data for modeling, I split the dataset into training and test subsets.

SVC: * Prior to instantiating my first SVC, I set the cost and gamma hyper parameters. I set the cost hyper parameter to 0.9 which is quite stringent and therefore set a hard margin right off of the bat. This means that there is a greater penalty with misclassifications. I set the gamma hyper parameter equal to 5, this value defines how influential the feature differences are in the model. This first SVC used the linear kernel and the f1-score was 0.76, a pretty good score but it's possible that fine tuning the hyper parameters will make it better. * I started by trying out a few different kernels. If a linear kernel doesn't work as well because you're data is fairly non-linear you can try other kernels, such as, "rbf" or "poly" kernels to can transform your features to be more linear and improve the model. The results were slightly better with the rbf and poly kernels with f1-scores of 0.77 and 0.78, respectively. * I decided to move forward with the poly kernel since it had the highest f1-score. I tweaked the cost hyper parameter to be a bit more relaxed since we are just predicting whether a fish has greater than or less than the average number of rings in the dataset. This means that there will not be as large of a penalty for misclassifications. However, even with a cost of 0.6, the f1-score didn't change. Therefore, I concluded that the best SVC is a model with the poly kernel, cost = 0.9, and gamma = 5.

SVR: * To prepare the data for an SVR, I selected the same features as the SVC but this time used the continuous "rings" column as the target. * I fit an SVR with the poly kernel and set the cost = 0.8 to predict the number of rings an Abalone fish has based on their physical attributes. * I evaluated the model by quantifiying the R-squared, the mean squared error, and the variance. The R-squared was pretty weak with a value of 0.53 but the mean squared error and variance were relatively low.

Final Conclusions: * Based on my findings, it's pretty hard to predict the number of rings of an Abalone fish based on their physical attributes. However, the best approach I identified was

using an SVC with a poly kernel to predict whether the number of rings of the fish was above or below the average of the dataset.